

DRAFT ISO/IEC 14496-10 : 2002 (E)

7.3.2.9 RBSP slice trailing bits syntax

<code>rbsp_slice_trailing_bits() {</code>	Category	Descriptor
<code> rbsp_stop_bit /* equal to 1 */</code>	All	f(1)
<code> if(entropy_coding_mode == 1)</code>		
<code> while(next_bits(1) == '1')</code>		
<code> cabac_stuffing_bit /* equal to 1 */</code>	All	f(1)
<code> while(!byte_aligned())</code>		
<code> rbsp_alignment_bit /* equal to 0 */</code>	All	f(1)
<code>}</code>		

DRAFT ITU-T Rec. H.264 (2002 E)

19

7.3.3 Slice header syntax

slice_header() {	Category	Descriptor
pic_parameter_set_id	4	ue(v)
frame_num	4	u(v)
pic_structure	4	ue(v)
first_mb_in_slice	4	u(v)
slice_type_idc	4	ue(v)
if(pic_order_cnt_type == 0)		
pic_order_cnt	4	u(v)
else if(pic_order_cnt_type == 1)		
delta_pic_order_cnt	4	se(v)
if(redundant_slice_flag)		
redundant_pic_cnt	4	ue(v)
if(slice_type_idc == BiPred)		
direct_spatial_mv_pred_flag	4	u(1)
num_ref_idx_active_override_flag	4	u(1)
if(num_ref_idx_active_override_flag) {		
if(slice_type_idc == Pred slice_type_idc == SPred slice_type_idc == BiPred) {		
num_ref_idx_l0_active_minus1	4	ue(v)
if(slice_type_idc == BiPred)		
num_ref_idx_l1_active_minus1	4	ue(v)
}		
}		
ref_idx_reordering()	4	
if((weighted_pred_flag && ((slice_type_idc == Pred) (slice_type_idc == SPred)) (weighted_bipred_explicit_flag && (slice_type_idc == BiPred))))		
pred_weight_table()	4	
ref_pic_buffer_management()	4	
slice_qp_delta	4	se(v)
if(slice_type_idc == SPred slice_type_idc == SIIntra) {		
if(slice_type_idc == SPred)		
sp_for_switch_flag	4	u(1)
slice_qp_s_delta	4	se(v)
}		
if(send_filter_parameters_flag == 1) {		
disable_deblocking_filter_flag	4	u(1)
if(!disable_deblocking_filter_flag) {		
slice_alpha_c0_offset_div2	4	se(v)
slice_beta_offset_div2	4	se(v)
}		
}		
if(num_slice_groups_minus1 > 0 && mb_allocation_map_type >= 4 && mb_allocation_map_type <= 6)		
slice_group_change_cycle	4	u(v)
}		

DRAFT ISO/IEC 14496-10 : 2002 (E)

7.3.3.1 Reference index reordering syntax

ref_idx_reordering() {	Category	Descriptor
if(slice_type() != Intra && slice_type() != SIntra) {		
ref_idx_reordering_flag_l0	4	u(1)
if(ref_idx_reordering_flag_l0) {		
NumRefIdxList = 1		
do {		
remapping_of_pic_nums_idc	4	ue(v)
if(remapping_of_pic_nums_idc == 0 remapping_of_pic_nums_idc == 1)		
abs_diff_pic_num_minus1	4	ue(v)
else if(remapping_of_pic_nums_idc == 2)		
long_term_pic_idx	4	ue(v)
} while(remapping_of_pic_nums_idc != 3)		
}		
}		
if(slice_type() == BiPred) {		
ref_idx_reordering_flag_l1	4	u(1)
if(ref_idx_reordering_flag_l1) {		
do {		
remapping_of_pic_nums_idc	4	ue(v)
if(remapping_of_pic_nums_idc == 0 remapping_of_pic_nums_idc == 1)		
abs_diff_pic_num_minus1	4	ue(v)
else if(remapping_of_pic_nums_idc == 2)		
long_term_pic_idx	4	ue(v)
} while(remapping_of_pic_nums_idc != 3)		
}		
}		
}		

DRAFT ITU-T Rec. H.264 (2002 E)

21

7.3.3.2 Prediction weight table syntax

<code>pred_weight_table() {</code>	Category	Descriptor
<code>luma_log_weight_denom</code>	4	ue(v)
<code>chroma_log_weight_denom</code>	4	ue(v)
<code>for(i = 0; i <=</code> <code>num_ref_idx_l0_active_minus1; i++) {</code>		
<code>luma_weight_flag_l0</code>	4	u(1)
<code>if(luma_weight_flag_l0) {</code>		
<code>luma_weight_l0[i]</code>	4	se(v)
<code>luma_offset_l0[i]</code>	4	se(v)
<code>}</code>		
<code>chroma_weight_flag_l0</code>	4	u(1)
<code>if(chroma_weight_flag_l0)</code>		
<code>for(j = 0; j < 2; j++) {</code>		
<code>chroma_weight_l0[i][j]</code>	4	se(v)
<code>chroma_offset_l0[i][j]</code>	4	se(v)
<code>}</code>		
<code>}</code>		
<code>if(slice_type() == BiPred) {</code>		
<code>for(i = 0; i <=</code> <code>num_ref_idx_l1_active_minus1; i++) {</code>		
<code>luma_weight_flag_l1</code>	4	u(1)
<code>if(luma_weight_flag_l1) {</code>		
<code>luma_weight_l1[i]</code>	4	se(v)
<code>luma_offset_l1[i]</code>	4	se(v)
<code>}</code>		
<code>chroma_weight_flag_l1</code>	4	u(1)
<code>if(chroma_weight_flag_l1)</code>		
<code>for(j = 0; j < 2; j++) {</code>		
<code>chroma_weight_l1[i][j]</code>	4	se(v)
<code>chroma_offset_l1[i][j]</code>	4	se(v)
<code>}</code>		
<code>}</code>		
<code>num_custom_bipred_weights</code>	4	ue(v)
<code>for(i = 0; i < num_custom_bipred_weights; i++) {</code>		
<code>if(num_ref_idx_l0_active_minus1</code> <code>num_ref_idx_active_l0_minus1 > 0)</code>		
<code>irp_l0</code>	4	xe(v)
<code>if(num_ref_idx_l1_active_minus1</code> <code>num_ref_idx_active_l1_minus1 > 0)</code>		
<code>irp_l1</code>	4	xe(v)
<code>luma_weight_bipred_l0[irp_l0][irp_l1]</code>	4	se(v)
<code>luma_weight_bipred_l1[irp_l0][irp_l1]</code>	4	se(v)
<code>luma_offset_bipred[irp_l0][irp_l1]</code>	4	se(v)
<code>chroma_weight_flag_bipred[irp_l0][irp_l1]</code>	4	u(1)
<code>if(chroma_weight_flag_bipred[irp_l0][irp_l1])</code>		
<code>for(j = 0; j < 2; j++) {</code>		
<code>chroma_weight_bipred_l0[irp_l0][irp_l1][j]</code>	4	se(v)

DRAFT ISO/IEC 14496-10 : 2002 (E)

chroma_weight_bipred_11 [irp_l0][irp_l1][j]	4	se(v)
chroma_offset_bipred [irp_l0][irp_l1][j]	4	se(v)
}		
}		
}		
}		

7.3.3.3 Reference picture buffer management syntax

ref_pic_buffer_management () {	Category	Descriptor
ref_pic_buffering_mode_flag	4 7	u(1)
if(ref_pic_buffering_mode == 1)		
do {		
memory_management_control_operation	4 7	ue(v)
if(memory_management_control_operation == 1 memory_management_control_operation == 3)		
difference_of_pic_nums_minus1	4 7	ue(v)
else-if(memory_management_control_operation == 2 memory_management_control_operation == 3)		
long_term_pic_idx	4 7	ue(v)
else-if(memory_management_control_operation == 4)		
max_long_term_pic_idx_plus1	4 7	ue(v)
} while(memory_management_control_operation != 0 && memory_management_control_operation != 5)		
}		

7.3.4 Slice data syntax

slice_data() {	Category	Descriptor
if(mb_frame_field_adaptive_flag && (pic_structure == 0 pic_structure == 3 pic_structure == 4)) {		
MbPairY = first_mb_in_slice / (pic_width_in_mb_minus1 + 1)		
MbPairX = first_mb_in_slice % (pic_width_in_mb_minus1 + 1)		
MbNum = (MbPairY << 1) * (pic_width_in_mb_minus1 + 1) - MbPairX		
} else		
__MbNum = first_mb_in_slice		
do {		
if(slice_type() != Intra && slice_type() != SIntra)		
if(entropy_coding_mode == 0) {		
mb_skip_run	4	ue(v)
MoreDataFlag = more_rbsp_data()		
} else {		
mb_skip_flag	4	ae(v)
MoreDataFlag = !mb_skip_flag		
}		
if(MoreDataFlag)		
if(mb_frame_field_adaptive_flag && (pic_structure == 0 pic_structure == 3 pic_structure == 4) && (slice_type() != BiPred)) {		
if(MbNum % 2 == 0)		
MbFieldDecodingFlag = 1		
if(MbFieldDecodingFlag) {		
mb_field_decoding_flag	4	u(1) ae(v)
MbFieldDecodingFlag = 0		
} else		
mb_field_decoding_flag = 0		
}		
if(adaptive_block_size_transform_flag == 0)		
macroblock_layer()	4 5 6	
else		
macroblock_layer_abt()	4 5 6	
if(entropy_coding_mode == 0)		
MoreDataFlag = more_rbsp_data()		
else {		
if(MbNum < num_mb_in_pic - MAX_MB_ADDRESS) {		
if(mb_frame_field_adaptive_flag && (pic_structure == 0 pic_structure == 3 pic_structure == 4) && MbNum % 2 != 0)		
MoreDataFlag = 1		
else {		
end_of_slice_flag	4	ae(v)
MoreDataFlag = !end_of_slice_flag		

DRAFT ISO/IEC 14496-10 : 2002 (E)

} else		
MoreDataFlag = 0		
}		
MbNum++		
} while(MoreDataFlag)		
}		

Formatted: Don't keep with next

{Ed. Note: Syntax may not be quite correct for MB-level APT. APT open issue.}

NOTE – macroblock_layer_abt() is specified in subclause 12.2.1.

7.3.5 Macroblock layer syntax

macroblock_layer() {	Category	Descriptor
mb_type	4	ue(v) ae(v)
if(num_mb_partition[mb_type] == 4)		
sub_mb_pred(mb_type)	4	
else		
mb_pred(mb_type)	4	
SendResidual = 0		
if(mb_partition_pred_mode(, 1) == Intra && mb_type != Intra 4x4)+ /* Intra 16x16 X Y Z mb_type */		
SendResidual = 1		
delta_qp	4	se(v) ae(v)
residual()	5 6	
} else {		
coded_block_pattern	4	me(v) ae(v)
if(coded_block_pattern > 0)+		
SendResidual = 1		
}		
if(SendResidual) {		
if('mb_frame_field_adaptive_flag' (mb_frame_field_adaptive_flag && (pic_structure == 0 pic_structure == 3 pic_structure == 4) && first_non_skip_mb_in_pair())		
delta_qp	4	se(v) ae(v)
residual()	5 6	
}		
}		
}		

{Ed. Note: Do not send delta_qp twice for the same macroblock pair when MB-level APT is in use. APT open issue.}

7.3.5.1 Macroblock prediction syntax

	Category	Descriptor
mb_pred (mb_type) {		
if(mb_partition_pred_mode(mb_type, 1) == Intra) {		
if(mb_type == Intra_4x4 mb_type == SIntra_4x4)		
for(i = 0; i < num_mb_intra_partition(mb_type); i++) /* for each 4x4 luma block */		
intra_pred_mode	4	ce(v) ae(v)
intra_chroma_pred_mode	4	ue(v) ae(v)
} else if(mb_type != Direct_16x16) {		
for(i = 0; i < num_mb_partition(mb_type); i++)		
if(num_ref_idx_l0_active_minus1 > 0 && mb_partition_pred_mode(mb_type, i) != Pred_L1)		
ref_idx_l0	4	ue(v) ae(v)
for(i = 0; i < num_mb_partition(mb_type); i++) {		
if(num_ref_idx_l1_active_minus1 > 0 && mb_partition_pred_mode(mb_type, i) != Pred_L0)		
ref_idx_l1	4	ue(v) ae(v)
for(i = 0; i < num_mb_partition(mb_type); i++) {		
if(mb_partition_pred_mode(mb_type, i) != Pred_L1)		
for(j = 0; j < 2; j++)		
mvd_l0 [i][j]	4	se(v) ae(v)
for(i = 0; i < num_mb_partition(mb_type); i++) {		
if(mb_partition_pred_mode(mb_type, i) != Pred_L0)		
for(j = 0; j < 2; j++)		
mvd_l1 [i][j]	4	se(v) ae(v)
}		
}		

DRAFT ISO/IEC 14496-10 : 2002 (E)

7.3.5.2 Sub macroblock prediction syntax

sub_mb_pred(mb_type) {	Category	Descriptor
for(i = 0; i < 4; i++) /* for each sub macroblock */		
sub_mb_type[i]	4	ue(v) ae(v)
IntraChromaPredModeFlag = 0		
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(sub_mb_type[i] == Intra_8x8)		
for(j = 0; j < num_sub_mb_intra_partition(sub_mb_type[i]);		
j++) /* num_sub_mb_intra_partition() = 4 */		
intra_pred_mode	4	cc(v) ae(v)
IntraChromaPredModeFlag = 1		
}		
if(IntraChromaPredModeFlag)		
intra_chroma_pred_mode	4	ue(v) ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(num_ref_idx_l0_active_minus1 > 0 &&		
mb_type != Pred_8x8ref0 &&		
sub_mb_type[i] != Intra_8x8 &&		
sub_mb_type[i] != Direct_8x8 &&		
sub_mb_pred_mode(sub_mb_type[i]) != Pred_L1)		
ref_idx_l0	4	ue(v) ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(num_ref_idx_l1_active_minus1 > 0 &&		
(sub_mb_type[i] != Intra_8x8 &&		
sub_mb_type[i] != Direct_8x8 &&		
sub_mb_pred_mode(sub_mb_type[i]) != Pred_L0)		
ref_idx_l1	4	ue(v) ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(sub_mb_type[i] != Intra_8x8 &&		
sub_mb_type[i] != Direct_8x8 &&		
sub_mb_pred_mode(sub_mb_type[i]) != Pred_L1)		
for(j = 0; j < num_sub_mb_partition(sub_mb_type[i]); j++)		
for(k = 0; k < 2; k++)		
mvd_l0[i][j][k]	4	se(v) ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(sub_mb_type[i] != Intra_8x8 &&		
sub_mb_type[i] != Direct_8x8 &&		
sub_mb_pred_mode(sub_mb_type[i]) != Pred_L0)		
for(j = 0; j < num_sub_mb_partition(sub_mb_type[i]); j++)		
for(k = 0; k < 2; k++)		
mvd_l1[i][j][k]	4	se(v) ae(v)
}		

DRAFT ITU-T Rec. H.264 (2002 E)

27

7.3.5.3 Residual data syntax

residual(mb_type) {	Category	Descriptor
if(entropy_coding_mode == 1)		
residual_4x4block = residual_4x4block_cabac()	5 6	
else		
residual_4x4block = residual_4x4block_cavlc()	5 6	
if(mb_type == Intra_16x16)		
residual_4x4block(intra16x16DC, 16)	5	
for(i8x8 = 0; i8x8 < 4; i8x8++) /* each luma 8x8 block */		
for(i4x4 = 0; i4x4 < num_sub_blocks(); i4x4++) /* each 4x4 sub-block of block */		
if(coded_block_pattern & (1 << i8x8))		
if(mb_type == Intra_16x16)		
residual_4x4block(intra16x16AC, 16)	5	
else		
residual_4x4block(luma, 16)	5 6	
if(coded_block_pattern & 0x30) /* chroma DC residual coded */		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
residual_4x4block(chromaDC, 4)	5 6	
if(coded_block_pattern & 0x20) /* chroma AC residual coded */		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
for(i4x4 = 0; i4x4 < 4; i4x4++)		
residual_4x4block(chromaAC, 16)	5 6	
}		

DRAFT ISO/IEC 14496-10 : 2002 (E)

Formatted: Default Paragraph Font

7.3.5.3.1 Residual 4x4 block CAVLC syntax

residual_4x4block_cavlc(block_mode, MaxNumCoeff) {	Category	Descriptor
coeff_token	5 6	ce(v)
NumCoeffs = total_coeff(coeff_token) - trailing_ones(coeff_token)		
if(trailing_ones(coeff_token) > 0)		
for(i = trailing_ones(coeff_token) - 1; i >= 0; i--)		
trailing_ones_sign[i]	5 6	u(1)
if(total_coeff(coeff_token) > 0) {		
for(i = NumCoeffs - 1; i >= 0; i--)		
coeff_level[i]	5 6	ce(v)
if(total_coeff(coeff_token) < MaxNumCoeff) {		
total_zeros	5 6	ce(v)
i = total_coeff(coeff_token) - 1		
ZerosLeft = total_zeros		
if(i > 0 && ZerosLeft > 0) {		
do {		
run_before[i]	5 6	ce(v)
ZerosLeft -= run_before[i]		
i--		
} while(ZerosLeft == 0 i == 0)		
run_before[i] = ZerosLeft		
}		
}		
}		

Formatted: Font: Bold

7.3.5.3.2 Residual 4x4 block CABAC syntax

residual_4x4block_cabac(BlockType, MaxNumCoeff) {	Category	Descriptor
coded_block_flag	5 6	ae(v)
if(coded_block_flag) {		
for(i = 0; i < MaxNumCoeff - 1; i++) {		
significant_coeff_flag[i]	5 6	ae(v)
if(significant_coeff_flag[i]) {		
last_significant_coeff_flag[i]	5 6	ae(v)
if(last_significant_coeff_flag[i])		
MaxNumCoeff = i + 1		
}		
}		
coeff_absolute_value_minus1[MaxNumCoeff - 1]	5 6	ae(v)
coeff_sign[MaxNumCoeff - 1]	5 6	ae(v)
for(i = MaxNumCoeff - 2; i >= 0; i--)		
if(significant_coeff_flag[i]) {		
coeff_absolute_value_minus1[i]	5 6	ae(v)
coeff_sign[i]	5 6	ae(v)
}		
}		
}		

DRAFT ITU-T Rec. H.264 (2002 E)

29

7.4 Semantics

7.4.1 NAL unit semantics

NOTE - The Video Coding Layer (VCL) is specified to efficiently represent the content of the video data. The Network Abstraction Layer (NAL) is specified to format that data and provide header information in a manner appropriate for conveyance by the transport layers or storage media. All data are contained in NAL units, each of which contains an integer number of bytes. A NAL unit specifies a generic format for use in both packet-oriented and bitstream systems. The format of NAL units for both packet-oriented transport and bitstream is identical except that each NAL unit can be preceded by a start code prefix in a bitstream-oriented transport layer.

NumBytesInNALunit specifies the size of the NAL unit in bytes. This value is required for decoding of the NAL unit and shall be conveyed by external means. Framing of NAL units is necessary to enable inference of NumBytesInNALunit. Such framing into a byte stream format is specified in Annex B and other methods for framing may be specified outside of this Recommendation | International Standard.

NOTE - Any sequence of bits can be formatted into a sequence of bytes in a manner defined as an RBSP by suffixing the data with `rbp_trailing_bits()`, and any RBSP can be encapsulated in a NAL unit a manner that prevents emulation of byte stream start code prefixes within the NAL unit.

forbidden_bit shall be zero.

NOTE - The forbidden_bit may be used by external specifications to signal potentially corrupt NAL units.

nal_storage_idc equal to 0 signals that the content of the NAL unit belongs either to a picture that is not stored in the reference picture buffer, SEI data or Filler data. **nal_storage_idc** shall not be 0 for sequence parameter set or picture parameter set NAL units. If **nal_storage_idc** is 0 for one slice or data partition NAL unit of a particular picture, it shall be 0 for all slice and data partition NAL units of the picture. **nal_storage_idc** greater than 0 signals that the content of the NAL unit belongs to a decoded picture that is stored in the reference picture buffer.

NOTE - In addition to signalling non-stored content, external specifications may use **nal_storage_idc** to indicate the relative transport priority of the NAL unit in a manner not specified in this Recommendation | International Standard. The value 0 should be used to signal the lowest transport priority and the priority should grow in ascending order of **nal_storage_idc** values.

DRAFT ISO/IEC 14496-10 : 2002 (E)

nal_unit_type indicates the type of element contained in the NAL unit according to the types specified in Table 7-1.

Table 7-1 – NAL Unit Type Codes

Value of nal_unit_type	Content of NAL unit and RBSP syntax structure	Category
0x0	Reserved for external use	
0x1	Coded slice slice_layer_no_partitioning_rbsp()	4, 5, 6
0x2	Coded data partition A (DPA) dpa_layer_rbsp()	4
0x3	Coded data partition B (DPB) dpb_layer_rbsp()	5
0x4	Coded data partition C (DPC) dpc_layer_rbsp()	6
0x5	Coded slice of an IDR picture slice_layer_no_partitioning_rbsp()	4, 5
0x6	Supplemental Enhancement Information (SEI) sei_rbsp()	7
0x7	Sequence Parameter Set (SPS) seq_parameter_set_rbsp()	0
0x8	Picture Parameter Set (PPS) pic_parameter_set_rbsp()	1
0x9	Picture Delimiter (PD) pic_delimiter_rbsp()	8
0xA	Filler Data (FD) filler_data_rbsp()	9
0xB – 0x17	Reserved	
0x18 – 0x1F	For external use	

An instantaneous decoder refresh picture (IDR picture) implies that all pictures in the multi-picture buffer are marked as “unused” except the current picture. Moreover, the maximum long-term index is reset to zero. An IDR picture contains only I or SI slices, and IDR slice type shall be used for all slices of an IDR picture.

rbbsp[i] a raw byte sequence payload is defined as an ordered sequence of bytes that contains an SODB. The RBSP contains the SODB in the following form:

- a) If the SODB is null, the RBSP is also null.
- b) Otherwise, the RBSP shall contain the SODB in the following form:
 - 1) The first byte of the RBSP shall contain the (most significant, left-most) eight bits of the SODB; the next byte of the RBSP shall contain the next eight bits of the SODB, etc.; until fewer than eight bits of the SODB remain.
 - 2) The final byte of the RBSP shall have the following form:
 - i) The first (most significant, left-most) bits of the final RBSP byte shall contain the remaining bits of the SODB, if any,
 - ii) The next bit of the final RBSP byte shall consist of a single **rbbsp_stop_bit** having the value one ('1'), and
 - iii) Any remaining bits of the final RBSP byte, if any, shall consist of one or more **rbbsp_alignment_bit** having the value zero ('0').

The last byte of a RBSP shall never have the value zero (0x00), because it contains the **rbbsp_stop_bit**.

If the boundaries of the RBSP are known, the decoder can extract the SODB from the RBSP by concatenating the bits of the bytes of the RBSP and discarding the **rbbsp_stop_bit**, which is the last (least significant, right-most) bit having the value one ('1'), and discarding any following (less significant, farther to the right) bits that follow it, which have the value zero ('0').

Syntax structures having these RBSP properties are denoted in the syntax tables using an "rbsp" suffix. These structures shall be carried within NAL units as the content of the rbsp[i] data bytes. The association of the RBSP syntax structures to the NAL units shall be as specified in Table 7-1.

emulation_prevention_byte is a byte equal to 0x03.

Within the NAL unit, an **emulation_prevention_byte** shall be present after an rbsp[i] byte having the value zero (0x00) if and only if a next byte of RBSP data rbsp[i+1] follows that has one of the following four values:

- zero (0x00)
- one (0x01)
- two (0x02)
- three (0x03)

NOTE - Example encoder procedure. The encapsulation of an SODB within an RBSP and the encapsulation of an RBSP within a NAL unit is specified to prevent the emulation of start codes within NAL units while allowing any arbitrary SODB to be represented within a NAL unit and enable identification of the end of the SODB within the NAL unit.

The encoder can produce a NAL unit from an RBSP by the following procedure:

The RBSP data is searched for byte-aligned bits of the following binary patterns:

'00000000 000000xx' (where xx represents any 2 bit pattern: 00, 01, 10 or 11),

and a byte having the value three (0x03) is inserted to replace these bit patterns with the patterns

'00000000 00000011 000000xx'

The resulting sequence of bytes is then prefixed with the first byte of the NAL unit containing the indication of the type of RBSP data structure it contains.

This process can allow any RBSP data to be sent in NAL unit while ensuring that no long SCP and no byte-aligned short SCP is emulated in the NAL unit.

7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics

7.4.2.1 Sequence parameter set RBSP semantics

A sequence parameter set is called an active sequence parameter set when an IDR NAL unit refers to it. The parameters of an active sequence parameter set shall be replaced only when an IDR NAL unit refers to a different sequence parameter set.

A picture parameter set includes the parameters that remain unchanged within a coded picture. Every picture parameter set shall refer to the active sequence parameter set. A decoded picture parameter set is an active picture parameter set when the first slice NAL unit or first DPA NAL unit of a coded picture refers to it. The picture parameters of an active picture parameter set shall be replaced only when the first slice NAL unit or DPA NAL unit of a picture refers to a different picture parameter set.

NOTE - The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. The sequence parameter set, the picture parameter set, and the slice header contain all the parameters needed to decode the slice data. It is recommended to convey sequence and picture parameter sets out-of-band using a reliable transport mechanism. However, if an application requires a self-contained bitstream, in-band parameter set information units may be used. In error-prone transmission environments, in-band sequence and picture parameter set information units should be protected in a way that assures their successful reception. Synchronization between in-band and out-of-band transmission of the sequence and picture parameter set information is outside of the scope of this Recommendation | International Standard.

profile_idc and **level_idc** indicate profile and level as specified in Annex A.

seq_parameter_set_id identifies the sequence parameter set to be referred.

log2_max_frame_num_minus4 specifies the MAX_FN used in frame number related arithmetic as follows:

$$\text{MAX_FN} = 2^{(\text{log2_max_frame_num_minus4} + 4)} \quad (7-1)$$

The value of log2_max_frame_num_minus4 shall be in the range of 0 to 12, inclusive.

pic_order_cnt_type equal to 0 or 1 indicates the method to code picture order count (see subclause 8.3.2). pic_order_cnt_type values greater than 1 are reserved.

log2_max_pic_order_cnt_minus4 is used to specify the MAX_PIC_ORDER_CNT used in picture order count related arithmetic as follows:

$$\text{MAX_PIC_ORDER_CNT} = 2^{(\text{log2_max_pic_order_cnt_minus4} + 4)} \quad (7-2)$$

DRAFT ISO/IEC 14496-10 : 2002 (E)

The size of the `pic_order_cnt` parameter in the slice header is $\log_2 \text{max_pic_order_cnt_minus4} + 4$ bits. The value of $\log_2 \text{max_pic_order_cnt_minus4}$ shall be in the range of 0 to 12, inclusive.

offset_for_non_stored_pic indicates an expected picture order count difference of a non-stored picture compared to a stored picture having the same `frame_num` as the non-stored picture.

num_stored_frames_in_pic_order_cnt_cycle signals the number of frame numbers in a picture order count cycle. A picture order count cycle is a repetitive pattern of picture order count differences, each of which corresponds to a `frame_num` increment of one.

offset_for_stored_frame indicates an expected difference of picture order count corresponding to a `frame_num` increment of one. Notation $\text{offset_for_stored_frame}_n$ indicates the `offset_for_stored_frame` corresponding to index n in the picture order count cycle. $\text{offset_for_stored_frame_in_pic_order_cnt_cycle}$ is the sum of all values of `offset_for_stored_frame`.

num_of_ref_frames specifies the total number of short- and long-term pictures in the reference picture buffer.

If **required_frame_num_update_behaviour** equal to 1 specifies a specific decoder behaviour in case of missing frame numbers.

picframe_width_in_mbs_minus1 and **picframe_height_in_mbs_minus1** specify the size of the luma frame-picture array internal to the decoder in units of macroblocks. The **frame-picture_width** and **height** in units of macroblocks is computed by adding 1 to the decoded values of each of these parameters. The maximum macroblock address, `MAX_MB_ADDRESS`, shall be calculated according to Equation 7-3.

$$\text{MAX_MB_ADDRESS} = (\text{picframe_width_in_mbs_minus1} + 1) \times (\text{picframe_height_in_mbs_minus1} + 1) - 1 \quad (7-3)$$

[Ed. Note: Fix for MB pair address: divide height by two in equation with proper rounding. AFF open issue.]

The `NUM_BITS_IN_MB_ADDRESS` indicates the number of bits used to code a macroblock address with a fixed-length unsigned integer. It is calculated as follows:

If a picture is a field-structured picture or if macroblock-adaptive frame/field coding is not in use for the picture, then

$$\text{NUM_BITS_IN_MB_ADDRESS} = \text{Ceil}(\text{Log2}(\text{MAX_MB_ADDRESS} + 1)) \quad (7-4)$$

otherwise, `NUM_BITS_IN_MB_ADDRESS` shall be

$$\text{NUM_BITS_IN_MB_ADDRESS} = \text{Ceil}(\text{Log2}(\text{MAX_MB_ADDRESS} + 1) - 1) \quad (7-4a)$$

send_filter_parameters_flag specifies whether a set of parameters controlling the characteristics of the deblocking filter is indicated in the slice header.

constrained_intra_pred_flag equal to zero indicates normal intra prediction, whereas one indicates constrained intra prediction, where no intra prediction is done from macroblocks coded with `mb_pred_type != Intra`.

mb_frame_field_adaptive_flag equal to zero indicates no switching between frame and field decoding mode at the macroblock layer, whereas one indicates the use of switching between frame and field decoding mode at the macroblock layer.

vui_seq_parameters_flag equal to zero specifies that default parameter values for the vui sequence parameters shall be applied.

7.4.2.2 Picture parameter set RBSP semantics

pic_parameter_set_id the picture parameter set identifier to be used for reference.

seq_parameter_set_id refers to the sequence parameter set that is used with this picture parameter set.

entropy_coding_mode equal to zero indicates VLC and CAVLC (see subclause 9.1), whereas value one indicates CABAC (see subclause 9.2). If CABAC is indicated, the `ae(v)` entropy coding is used for the assigned syntax elements.

motion_resolution equal to zero indicates 1/4 luma sample accurate motion resolution, and equal to one indicates 1/8 luma sample accurate motion resolution.

adaptive_block_size_transform_flag equal to zero indicates usage of 4x4 transforms for the luma residual, and equal one indicates usage of transforms of size 4x4, 4x8, 8x4, and 8x8 for the luma residual. Clause 12 specifies modifications as indicated in clause 7 that are related to syntax, semantics, and decoding process.

Formatted: Equation, Indent: Left: 1"

Formatted: Font: Not Bold

num_slice_groups_minus1 the number of slice groups is equal to **num_slice_groups_minus1** + 1. If **num_slice_groups_minus1** is zero, all slices of the picture belong to the same slice group.

NOTE – One slice group means that no flexible macroblock ordering is applied. If **num_slice_groups_minus1** is greater than zero, flexible macroblock ordering is in use.

mb_allocation_map_type the macroblock allocation map type is present only if **num_slice_groups_minus1** is greater than 0. This parameter indicates how the macroblock allocation map is coded. The value of this syntax element shall be in the range of 0 to 6, inclusive.

mb_allocation_map_type 0 is used to indicate interleaved slices.

mb_allocation_map_type 1 is used to indicate a dispersed macroblock allocation.

mb_allocation_map_type 2 is used to explicitly assign a slice group to each macroblock location in raster scan order.

mb_allocation_map_type 3 is used to indicate one or more “foreground” slice groups and a “leftover” slice group.

mb_allocation_map_types 4, 5 and 6 are used to indicate changing slice groups. **num_slice_groups_minus1** shall be 1, when **mb_allocation_map_type** is 4, 5 or 6.

If **mb_allocation_map_type** is 0, the **run_length** syntax element follows for each slice group. It indicates the number of consecutive macroblocks that are assigned to the slice group in raster scan order. After the macroblocks of the last slice group have been assigned, the process begins again from the first slice group. The process ends when all the macroblocks of a picture have been assigned.

NOTE - Example: To signal macroblock row interleaving in a QCIF picture (where all even numbered macroblocks are in slice group 0, and all odd numbered macroblocks are in slice group 1), the number of slice groups is two and **run_length** is 11 for both slice groups.

If **mb_allocation_map_type** is 1, the macroblock allocation map is formed using the following formula, where *n* is the number of columns in the picture (in macroblocks) and *p* is the number of slice groups to be coded. Specifically, macroblock position *x* is assigned to slice group *S* according to Equation 7-5.

$$S(x) = ((x \% n) + ((\text{floor}(x / n) * p) / 2)) \% p \quad (7-5)$$

If **mb_allocation_map_type** is 2, **slice_group_id** identifies a slice group of a macroblock.

NOTE - **slice_group_id** is repeated as often as there are macroblocks in the picture.

If **mb_allocation_map_type** is 3, **top_left_mb** and **bottom_right_mb** are specified for each **slice_group_id** except for the last one. The **top_left_mb** specifies the top-left corner of a rectangle and **bottom_right_mb** specifies the bottom-right corner. **top_left_mb** and **bottom_right_mb** are indicated as macroblock addresses. The foreground slice group contains the macroblocks that are within the indicated rectangle and that do not belong to any slice group having a smaller **slice_group_id**. The last **slice_group_id** is dedicated for the leftover slice group, which contains the macroblocks that are not covered by the foreground slice groups. The leftover slice group shall not be empty. The size of the **top_left_mb** and **bottom_right_mb** parameters **NUM_BITS_IN_MB_ADDRESS**.

If **mb_allocation_map_type** is 4, 5 or 6, **mb_allocation_map_type** and **slice_group_change_direction** indicate the refined macroblock allocation map type according to Table 7-2. The macroblock allocation map is generated each time the decoder starts decoding of a new picture as described in subclause 8.3.4.

Table 7-2– Refined macroblock allocation map type

mb_allocation_map_type	slice_group_change_direction	refined macroblock allocation map type
4	0	Box-out clockwise
4	1	Box-out counter-clockwise
5	0	Raster scan
5	1	Reverse raster scan
6	0	Wipe right
6	1	Wipe left

slice_group_change_rate_minus1 is the minimum non-zero number of macroblocks by which the size a slice group can change from one picture to the next. The **SLICE_GROUP_CHANGE_RATE** variable is defined as follows:

$$\text{SLICE_GROUP_CHANGE_RATE} = \text{slice_group_change_rate_minus1} + 1 \quad (7-6)$$

DRAFT ISO/IEC 14496-10 : 2002 (E)

The decoded value of `slice_group_change_rate_minus1` shall be in the range of 0 to `MAX_MB_ADDRESS - 1`, inclusive.

`num_ref_idx_l0_active_minus1` specifies the number of reference pictures minus 1 in the reference list 0 that are used to decode the picture.

`num_ref_idx_l1_active_minus1` specifies the number of reference pictures minus 1 in the reference list 1 that are used to decode the picture.

`weighted_pred_flag` equal to zero indicates that weighted prediction is not applied to P and SP slices. `weighted_pred_flag` equal to one indicates that weighted prediction is applied to P and SP slices.

`weighted_bipred_explicit_flag` equal to zero indicates that explicit weighted prediction is not applied to B slices. `weighted_bipred_explicit_flag` equal to one indicates that explicit weighted prediction is applied to B slices. `weighted_bipred_explicit_flag` shall be zero if `weighted_bipred_implicit_flag` is one.

`weighted_bipred_implicit_flag` equal to zero indicates that implicit weighted prediction is not applied to B slices. `weighted_bipred_implicit_flag` equal to one indicates that implicit weighted prediction is applied to B slices. `weighted_bipred_implicit_flag` shall be zero if `weighted_bipred_explicit_flag` is one.

`slice_qp_minus26` specifies the value of the default QP_Y for the macroblocks in an I, SI, P, SP, or B slice as specified in Equation 7-7. The value of this syntax element shall be in the range of -26 to +25, inclusive.

`slice_qp_s_minus26` specifies the value of the default QS_Y for the macroblocks in a SP or SI slice as specified in Equation 7-8. The value of this syntax element shall be in the range of -26 to +25, inclusive.

`redundant_slice_flag` indicates the presence of the `redundant_pic_cnt` parameter in all slice headers referencing the picture parameter set.

`vui_pic_parameters_flag` equal to zero specifies that default parameter values for the vui picture parameters shall be applied.

7.4.2.3 Supplemental enhancement information RBSP semantics

Supplemental Enhancement Information (SEI) contains information that is not necessary to decode VCI data correctly but is helpful for decoding or presentation purposes.

7.4.2.3.1 Supplemental enhancement information message semantics

An SEI NAL unit contains one or more SEI messages. Each SEI message consists of SEI header and SEI payload. The type and size of the SEI payload are coded using an extensible syntax. The SEI payload size is indicated in bytes. SEI payload types are specified in Annex C.

The SEI payload may have a SEI payload header. For example, a payload header may indicate to which picture the particular data belongs. The payload header shall be defined for each payload type separately.

An SEI message is associated with the next slice or data partition RBSP in decoding order.

`byte_ff` is a byte equal to 0xFF identifying a need for a longer representation of the syntax structure it is used within.

`last_payload_type_byte` identifies the payload type of the last entry in an SEI message.

`last_payload_size_byte` identifies the size of the last entry in an SEI message.

7.4.2.4 Picture delimiter RBSP semantics

The picture delimiter may be used to signal the boundary between pictures, i.e., if present, it shall be inserted before the first NAL unit of a picture in decoding order. There is no normative decoder process associated with the picture delimiter.

`pic_type` signals which slice coding types are used in the following picture in decoding order. Table 7-3 shows the slice coding types that can occur in a picture with a given `pic_type`.

Table 7-3— Meaning of `pic_type`

<code>pic_type</code>	Allowed slice_type_idc
000	Intra
001	Intra, Pred
010	Intra, Pred, BiPred

011	SIIntra
100	SIIntra, SPred
101	Intra, SIIntra
110	Intra, SIIntra, Pred, SPred
111	Intra, SIIntra, Pred, SPred, BiPred

If `adaptive_block_size_transform_flag == 1`, `pic_type == '000'`, `pic_type == '001'`, and `pic_type == '010'` are allowed.

non_stored_content_flag equal to 1 indicates that the picture is not stored in the reference picture buffer.

7.4.2.5 Filler data RBSP semantics

The filler data RBSP contains bytes whose value shall be equal to 0xFF.

7.4.2.6 Slice layer RBSP semantics

The slice layer RBSP consists of a slice header and slice data.

7.4.2.7 Data partition RBSP semantics

7.4.2.7.1 Data partition A RBSP semantics

When data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Partition A contains all symbols of Category 4.

NOTE - Category 4 consists of the header symbols of all coded MBs.

slice_id Each slice of a picture is associated a unique slice identifier within the picture. The first coded slice of the picture shall have identifier 0 and the identifier shall be incremented by one per each coded slice.

7.4.2.7.2 Data partition B RBSP semantics

When data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Data partition B contains all symbols of Category 5.

NOTE - Category 5 consists of the intra coded block patterns and coefficients.

slice_id Each slice of a picture is associated a unique slice identifier within the picture. The first coded slice of the picture shall have identifier 0 and the identifier shall be incremented by one per each coded slice.

7.4.2.7.3 Data partition C RBSP semantics

When data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Data partition C contains all symbols of Category 6.

NOTE - Category 6 consists of the inter coded block patterns and coefficients.

slice_id Each slice of a picture is associated a unique slice identifier within the picture. The first coded slice of the picture shall have identifier 0 and the identifier shall be incremented by one per each coded slice.

7.3.2.8 RBSP trailing bits semantics

rbsp_stop_bit is a single bit having the value one ('1').

rbsp_alignment_bit is a single bit having the value zero ('0').

7.3.2.9 RBSP slice trailing bits semantics

rbsp_stop_bit has the same semantics as in subclause 7.3.2.8.

rbsp_alignment_bit has the same semantics as in subclause 7.3.2.8.

cabac_stuffing_bit is a single bit having the value one ('1'). When `entropy_coding_mode` is equal to 1, the number of bins resulting from decoding the contents of the slice layer NAL unit shall not exceed $0.5 \times \text{NumBytesInNALunit}$. A number of inserted `cabac_stuffing_bit` guarantee this condition.

NOTE – The method for determining the number of inserted `cabac_stuffing_bit` at the encoder is as follows: First, at the beginning of the slice encoding, the index `CodeLength` pointing to the current position of the bit stream is stored in a variable `CodeLengthStored`, i.e., `CodeLength ← CodeLengthStored`. In addition, two counters `EBins` and `EBinsx8` are set to zero. Then, each time a symbol is encoded, `EBins` is incremented by one, i.e., `EBins ← EBins + 1`. In the renormalization procedure, each time a new byte of compressed data is written, the following procedure is done:

```
while(EBins > 7) {
    EBins ← EBins - 8
```

DRAFT ISO/IEC 14496-10 : 2002 (E)

```
EBinsx8 ← EBinsx8+1
```

After terminating the arithmetic encoding process such that the last pending bits have been written to the bitstream, the number of written bits is determined by $\text{CodeLength} \leftarrow 8 * (\text{CodeLength} - \text{CodeLengthStored})$. Given the number of encoded bins by $\text{EBins} \leftarrow 8 * \text{EBinsx8} + \text{EBins}$ and the number NumMbSlice of macroblocks per slice, the following procedure is done:

```
EBins ← 2*EBins
if (CodeLength >= 4*NumMbSlice)
{
  if (EBins > CodeLength)
    for(i = 0; i < EBins - CodeLength - 1; i++)
      putbits('1') /* writes cabac_stuffing_bit */
}
```

7.4.3 Slice header semantics

pic_parameter_set_id indicates the picture parameter set in use.

frame_num labels the frame. **frame_num** shall be incremented by 1 for each coded picture in decoding order, in modulo MAX_FN operation, relative to the **frame_num** of the previous stored frame in decoding order. The **frame_num** serves as a unique ID for each frame stored in the reference picture buffer. Therefore, a frame cannot be kept in the buffer after its **frame_num** has been used by another frame unless it has been assigned a long-term frame index as specified below. No **frame_num** of a frame to be added to the reference picture buffer shall equal to any other among the short-term frames in the reference picture buffer. A decoder which encounters a frame number on a current frame having a value equal to the frame number of some other short-term stored frame in the reference picture buffer should treat this condition as an error.

pic_structure identifies the picture structure according to Table 7-4.

Table 7-4 – Meaning of **pic_structure**

Value of pic_structure	Meaning
0	Progressive frame picture
1	Top field picture
2	Bottom field picture
3	Interlaced frame picture, whose top field precedes its bottom field in time.
4	Interlaced frame picture, whose bottom field precedes its top field in time.

Note that when top field and bottom field pictures are coded for a frame, the one that is decoded first is the one that occurs first in time.

first_mb_in_slice specifies the macroblock address of the first macroblock contained in this slice. The size of the **first_mb_in_slice** parameter is $\text{NUM_BITS_IN_MB_ADDRESS}$. The value of **first_mb_in_slice** shall be in the range of 0 to MAX_MB_ADDRESS , inclusive.

If macroblock-adaptive frame/field decoding is in use, **first_mb_in_slice** contains a macroblock pair address rather than a macroblock address and the number of macroblocks included in a slice shall be an even number. ~~[Ed. Note: AFF open issue.]~~

slice_type_idc indicates the coding type of the slice according to Table 7-5.

Table 7-5 – Meaning of **slice_type_idc**

Value of slice_type_idc	Prediction type of slice (slice type)
0	Pred (P slice)
1	BiPred (B slice)
2	Intra (I slice)
3	SPred (SP slice)
4	SIIntra (SI slice)

Table 7-6 specifies, which macroblock prediction types are allowed when a slice type is decoded.

Table 7-6 – Allowed macroblock prediction types for slice_type_idc

Prediction type of slice (slice type)	Allowed macroblock prediction type
Pred (P slice)	Intra, Pred
BiPred (B slice)	Intra, Pred, BiPred
Intra (I slice)	Intra
SPred (SP slice)	SPred, Intra
SIIntra (SI slice)	SIIntra, Intra

If `adaptive_block_size_transform_flag` == 1, the use of SI slices and SP slices is not allowed.

pic_order_cnt carries the picture order count coded in modulo `MAX_PIC_ORDER_CNT` arithmetic. An IDR picture shall have `pic_order_cnt` equal to 0. The size of the `pic_order_cnt` parameter is $\log_2 \text{max_pic_order_cnt_minus4} + 4$ bits.

delta_pic_order_cnt signals the picture order count difference compared to the offset picture order count as described in subclause 8.3.2.

redundant_pic_cnt is 0 for coded slices and data partitions belonging to the primary representation of the picture contents. The `redundant_pic_cnt` is greater than 0 for coded slices and data partitions that contain redundant coded representation of the picture contents. There should be no noticeable difference between the co-located areas of the decoded primary representation of the picture and any decoded redundant slices. Decoded slices having the same `redundant_pic_cnt` shall not overlap. Decoded slices having a `redundant_pic_cnt` greater than 0 may not cover the entire picture area.

direct_spatial_mv_pred_flag specifies the method used in the decoding process to determine the prediction values direct prediction. If `direct_spatial_mv_pred_flag` is set to 0, then direct mode motion parameters are calculated from the picture order count as described in subclause 10.3.3.2. Otherwise, if this flag is set to 1, then direct mode motion parameters are calculated using the spatial motion vector prediction technique as described in subclause 10.3.3.1.

num_ref_idx_active_override_flag equal to zero indicates that the `num_ref_idx_l0_active_minus1` and `num_ref_idx_l1_active_minus1` specified in the referred picture parameter set are in effect. `num_ref_idx_active_override_flag` equal to one indicates that the `num_ref_idx_l0_active_minus1` and `num_ref_idx_l1_active_minus1` specified in the referred picture parameter set are overridden by the following values in the slice header.

num_ref_idx_l0_active_minus1 specifies the number of reference pictures minus 1 in the reference picture list 0 that are used to decode the picture.

num_ref_idx_l1_active_minus1 specifies the number of reference pictures minus 1 in the reference picture list 1 that are used to decode the picture.

slice_qp_delta specifies the value of the QP_Y for the MBs in the slice unless modified by the value of `delta_qp` in the macroblock layer. From this value, the initial QP_Y parameter for the slice is computed as:

$$QP_Y = 26 + \text{slice_qp_minus26} + \text{slice_qp_delta} \quad (7-7)$$

The initial decoded QP_Y parameter shall be in the range of 0 to 51, inclusive. The value of QP_Y is initialised to the above result and this value is used for the decoding of each macroblock in the slice unless updated by a `delta_qp` sent in the macroblock layer.

sp_for_switch_flag indicates the decoding process to be used to decode the SP slice.

slice_qp_s_delta is signalled for SP and SI slices. The QS_Y parameter for the slice is computed as:

$$QS_Y = 26 + \text{slice_qp_s_minus26} + \text{slice_qp_s_delta} \quad (7-8)$$

The value of QS_Y shall be in the range of 0 to 51, inclusive. This value of QS_Y is used for the decoding of all macroblocks in the slice.

disable_deblocking_filter_flag equal to zero specifies that the deblocking filter shall be applied to the edges controlled by the macroblocks within the current slice. If `disable_deblocking_filter_flag` is 1, `Filter_Offset_A` and `Filter_Offset_B` shall both be inferred to be equal to -51. If not present in the slice header the value of this field shall be inferred to be zero.

DRAFT ISO/IEC 14496-10 : 2002 (E)

slice_alpha_c0_offset_div2 specifies the offset used in accessing the ALPHA and C0 deblocking filter tables for filtering operations controlled by the macroblocks within the slice. The decoded value of this parameter shall be in the range from -6 to +6, inclusive. From this value, the offset that shall be applied when addressing these tables is computed as:

$$\text{Filter_Offset_A} = \text{slice_alpha_c0_offset_div2} \ll 1$$

If not present in the slice header the value of this field shall be inferred to be zero unless **disable_deblocking_filter_flag** is 1.

slice_beta_offset_div2 specifies the offset used in accessing the BETA deblocking filter table for filtering operations controlled by the macroblocks within the slice. The decoded value of this parameter shall be in the range from -6 to +6, inclusive. From this value, the offset that is applied when addressing the BETA table of the deblocking filter is computed as:

$$\text{Filter_Offset_B} = \text{slice_beta_offset_div2} \ll 1$$

If not present in the slice header the value of this field shall be inferred to be zero unless **disable_deblocking_filter_flag** is 1.

slice_group_change_cycle \times **SLICE_GROUP_CHANGE_RATE** indicates the number of macroblocks in slice group 0. The size of the **slice_group_change_cycle** field is $\text{Ceil}(\text{Log2}(\text{Ceil}(\text{MAX_MB_LOCATION} + \text{SLICE_GROUP_CHANGE_RATE})))$. The maximum value of **slice_group_change_cycle** is $\text{Ceil}(\text{MAX_MB_LOCATION} + \text{SLICE_GROUP_CHANGE_RATE})$.

7.4.3.1 Reference index reordering semantics

The syntax elements **remapping_of_pic_nums_idc**, **abs_diff_pic_num_minus1**, and **long_term_pic_idx** specify the change from the default reference index lists to the reference index lists used for decoding the slice.

ref_idx_reordering_flag_l0 indicates whether the syntax elements **remapping_of_pic_nums_idc**, **abs_diff_pic_num_minus1**, and **long_term_pic_idx** are present for specifying the reference index list 0.

ref_idx_reordering_flag_l1 has the same semantics as **ref_idx_reordering_flag_l0** except that the reordering of the reference index list 1 is specified instead of the reference index list 0.

remapping_of_pic_nums_idc together with **abs_diff_pic_num_minus1** and **long_term_pic_idx** indicates which of the reference pictures are re-mapped. The restrictions and the mapping to the code number are specified in Table 7-7. The number of signalling **remapping_of_pic_nums_idc** is limited to the **num_ref_idx_l0_active_minus1** + 1.

Table 7-7 – **remapping_of_pic_nums_idc** operations for re-mapping of reference pictures

Value of remapping_of_pic_nums_idc	Re-mapping Specified
0	abs_diff_pic_num_minus1 is present and corresponds to a negative difference to add to a picture number prediction value
1	abs_diff_pic_num_minus1 is present and corresponds to a positive difference to add to a picture number prediction value
2	long_term_pic_idx is present and specifies the long-term index for a reference picture
3	End loop for re-mapping of reference picture set relative indexing default order

abs_diff_pic_num_minus1 plus 1 indicates the absolute difference between the picture number of the picture being remapped and the picture number prediction value.

long_term_pic_idx indicates the long-term picture index of the picture being remapped. In the case of frame-structured pictures, it shall be less than **max_long_term_pic_idx_plus1**; while in the case of field-structured pictures, it shall be less than $2 \times \text{max_long_term_pic_idx_plus1}$.

abs_diff_pic_num_minus1 **abs_diff_pic_num_minus1** + 1 corresponds to the absolute difference between picture numbers. [Ed. Note: specify operation when frame_num wraps at max_frame_num]

~~long_term_pic_idx~~ represents the long-term picture index to be re-mapped. It is restricted to the maximum number ~~max_long_term_pic_idx_plus1~~.

7.4.3.2 Reference picture buffer management semantics

The syntax elements ~~ref_pic_buffering_mode~~, ~~memory_management_control_operation~~, ~~difference_of_pic_nums_minus1~~, ~~long_term_pic_idx~~, and ~~max_long_term_pic_idx_plus1~~ specify the buffering of a stored decoded picture into the reference picture buffer. Further, the reference picture buffer can be modified by marking pictures as unused, by ~~assigned-assigning~~ long-term pictures indices ~~or-and~~ by ~~reset-resetting~~ of the reference picture buffer. The syntax elements ~~ref_pic_buffering_mode~~, ~~memory_management_control_operation~~, ~~difference_of_pic_nums_minus1~~, ~~long_term_pic_idx~~, and ~~max_long_term_pic_idx_plus1~~ shall be identical for all coded slices of a coded picture.

~~ref_pic_buffering_mode~~ specifies the buffering mode of the currently decoded picture and specifies how the reference picture buffer is modified after the current picture is decoded. The values for ~~ref_pic_buffering_mode~~ are specified in Table 7-8.

Table 7-8 – Interpretation of ~~ref_pic_buffering_mode~~

Value of ref_pic_buffering_mode	Reference picture buffering mode specified
0	Sliding Window-window buffering mode; A simple buffering mode providing a first-in first-out mechanism for pictures that are not assigned a long-term index
1	Adaptive buffering mode; A more flexible buffering mode than Sliding Window-window buffering mode providing syntax elements to specify marking pictures as unused, to assign long-term pictures indices, or-and to reset the reference picture buffer.

~~memory_management_control_operation~~ specifies a control operation to be applied to manage the reference picture buffer. The ~~memory_management_control_operation~~ parameter is followed by data necessary for the operation specified by the value of ~~memory_management_control_operation~~. ~~memory_management_control_operation~~ commands do not affect the buffer contents or the decoding process for the decoding of the current frame. They specify the necessary buffer status for the decoding of subsequent coded pictures. The values and control operations associated with ~~memory_management_control_operation~~ are defined in Table 7-9.

~~Note that memory_management_control_operation commands always indicate the same treatment for both fields of a frame, and the frame numbers referred to in memory_management_control_operation commands are always frame numbers even if the current picture is a field picture.~~

If ~~memory_management_control_operation~~ is Reset, all ~~frames-frames~~ and ~~fields~~ in the reference picture buffer (but not the current ~~frame-picture~~ unless specified separately) shall be marked “unused” (including both short-term and long-term ~~framespictures~~). Moreover, the maximum long-term ~~frame-picture~~ index shall be reset to zero.

The frame height and width shall not change within the bitstream except within a ~~frame-picture~~ containing a Reset ~~memory_management_control_operation~~ command.

A “stored ~~framepicture~~” shall not contain any ~~memory_management_control_operation~~ command which marks that (entire) ~~frame-picture~~ as “unused”. If the current ~~frame-picture~~ is non-stored ~~framepicture~~, the value of the ~~memory_management_control_operation~~ shall not contain any of the following types of ~~memory_management_control_operation~~ commands:

- A Reset ~~memory_management_control_operation~~ command,
- Any ~~memory_management_control_operation~~ command which marks any other ~~frame-picture~~ (other than the current ~~framepicture~~) as “unused” that has not also been marked as “unused” in the RPS layer of a prior stored ~~framepicture~~, or
- Any ~~memory_management_control_operation~~ command which assigns a long-term index to a ~~frame picture~~ that has not also been assigned the same long-term index in the RPS layer of a prior stored ~~framepicture~~.

Formatted: Normal

Table 7-9 – Memory management control operation (~~memory_management_control_operation~~) values

Value of	Memory Management Control Operation	Associated Data Fields Following
----------	-------------------------------------	----------------------------------

DRAFT ISO/IEC 14496-10 : 2002 (E)

memory_management_control_operation		
0	End memory_management_control_operation Loop	None (end of RPS layer)
1	Mark a Short-Term Frame picture as "Unused"	difference_of_pic_nums_minus1
2	Mark a Long-Term pFrame as "Unused"	long_term_pic_idx
3	Assign a Long-Term index to a Frame picture	difference of pic_nums and long_term_pic_idx
4	Specify the Maximum Long-Term Frame picture index	max_long_term_pic_idx_plus1
5	Reset	None

difference_of_pic_nums_minus1 is used to assign a long-term index to a picture or to mark a short-term picture as "unused".

long_term_pic_idx is used to assign a long-term index to a picture or to mark a long-term picture as "unused".

max_long_term_pic_idx_plus1 indicates the maximum index allowed for long-term reference frames (until receipt of another value of max_long_term_pic_idx_plus1). The decoder shall initially infer that max_long_term_pic_idx_plus1 is 0 until some other value has been received.

~~difference_of_pic_nums~~ is used in order to assign a long-term index to a frame or to mark a short-term frame as "unused".

~~long_term_pic_idx~~ follows ~~difference_of_pic_nums~~ if the operation is to assign a long-term index to a picture.

~~max_long_term_pic_idx_plus1~~ is used to determine the maximum index allowed for long-term reference frames (until receipt of another value of max_long_term_pic_idx_plus1). The decoder shall initially assume max_long_term_pic_idx_plus1 is 0 until some other value has been received. Upon receiving an max_long_term_pic_idx_plus1 parameter, the decoder shall consider all long-term frames having indices greater than the decoded value of max_long_term_pic_idx_plus1 - 1 as "unused" for referencing by the decoding process for subsequent frames. For all other frames in the reference picture buffer, no change of status shall be indicated by max_long_term_pic_idx_plus1.

7.4.3.3 Prediction weight table semantics

luma_log_weight_denom is the binary logarithm of the denominator for all luma weighting factors.

chroma_log_weight_denom is the binary logarithm of the denominator for all chroma weighting factors.

luma_weight_flag_l0 indicates whether weighting factors are present for the luma component of the list 0 prediction.

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for reference index i in list 0 prediction.

luma_offset_l0[i] is the additive offset applied to the luma prediction value for reference index i in list 0 prediction.

If luma_weight_flag_l0 is equal to zero, luma_weight_l0[i] shall be interpreted as equal to $2^{\text{luma_log_weight_denom}}$ and luma_offset_l0[i] shall be interpreted as equal to zero.

chroma_weight_flag_l0 indicates whether weighting factors are present for the Cb and Cr components of the list 0 prediction.

chroma_weight_l0[i][0] is the weighting factor applied to the Cb prediction values for reference index i in list 0 prediction.

chroma_offset_l0[i][0] is the additive offset applied to the Cb prediction values for reference index i in list 0 prediction.

chroma_weight_l0[i][1] is the weighting factor applied to the Cr prediction values for reference index i in list 0 prediction.

chroma_offset_l0[i][1] is the additive offset applied to the Cr prediction values for reference index i in list 0 prediction.

If `chroma_weight_flag_l0` is equal to zero, `chroma_weight_l0[i]` shall be interpreted as equal to $2^{\text{chroma_log_weight_denom}}$ and `chroma_offset_l0[i]` shall be interpreted as equal to zero.

`luma_weight_flag_l1` indicates whether weighting factors are present for the luma component of the list 1 prediction.

`luma_weight_l1[i]` is the weighting factor applied to the luma prediction values for reference index `i` in list 1 prediction.

`luma_offset_l1[i]` is the additive offset applied to the luma prediction value for reference index `i` in list 1 prediction.

If `luma_weight_flag_l1` is equal to zero, `luma_weight_l1[i]` shall be interpreted as equal to $2^{\text{luma_log_weight_denom}}$ and `luma_offset_l1[i]` shall be interpreted as equal to zero.

`chroma_weight_flag_l1` indicates whether weighting factors are present for the chroma component of the list 1 prediction.

`chroma_weight_l1[i][0]` is the weighting factor applied to the Cb prediction values for reference index `i` in list 1 prediction.

`chroma_offset_l1[i][0]` is the additive offset applied to the Cb prediction values for reference index `i` in list 1 prediction.

`chroma_weight_l1[i][1]` is the weighting factor applied to the Cr prediction values for reference index `i` in list 1 prediction.

`chroma_offset_l1[i][1]` is the additive offset applied to the Cr prediction values for reference index `i` in list 1 prediction.

If `chroma_weight_flag_l1` is equal to zero, `chroma_weight_l1[i][j]` shall be interpreted as equal to $2^{\text{chroma_log_weight_denom}}$ and `chroma_offset_l1[i][j]` shall be interpreted as equal to zero.

`num_custom_bipred_weights` is the number of custom weight and offset combinations sent for bi-predictive weighting.

`irp_l0` is the index of the reference picture in list 0 for which a custom weight and offset combination is indicated for custom bi-predictive weighting.

`irp_l1` is the index of the reference picture in list 1 for which a custom weight and offset combination is indicated for custom bi-predictive weighting.

`luma_weight_bipred_l0[irp_l0][irp_l1]` is the weighting factor applied to the luma prediction value for reference index `irp_l0` in list 0 when used with reference index `irp_l1` in list 1 for bi-prediction.

`luma_weight_bipred_l1[irp_l0][irp_l1]` is the weighting factor applied to the luma prediction value for reference index `irp_l1` in list 1 when used with reference index `irp_l0` in list 0 for bi-prediction.

`luma_offset_bipred[irp_l0][irp_l1]` is the additive offset applied to the luma prediction values when reference index `irp_l0` in list 0 is used with reference index `irp_l1` in list 1 for bi-prediction.

`chroma_weight_flag_bipred[irp_l0][irp_l1]` indicates whether custom weight and offset combinations are sent for Cb and Cr prediction when reference index `irp_l0` in list 0 is used with reference index `irp_l1` in list 1 for bi-prediction.

`chroma_weight_bipred_l0[irp_l0][irp_l1][0]` is the weighting factor applied to the Cb prediction value for reference index `irp_l0` in list 0 when used with reference index `irp_l1` in list 1 for bi-prediction.

`chroma_weight_bipred_l1[irp_l0][irp_l1][0]` is the weighting factor applied to the Cb prediction value for reference index `irp_l1` in list 1 when used with reference index `irp_l0` in list 0 for bi-prediction.

`chroma_offset_bipred[irp_l0][irp_l1][0]` is the additive offset applied to the Cb prediction values when reference index `irp_l0` in list 0 is used with reference index `irp_l1` in list 1 for bi-prediction.

`chroma_weight_bipred_l0[irp_l0][irp_l1][1]` is the weighting factor applied to the Cr prediction value for reference index `irp_l0` in list 0 when used with reference index `irp_l1` in list 1 for bi-prediction.

`chroma_weight_bipred_l1[irp_l0][irp_l1][1]` is the weighting factor applied to the Cr prediction value for reference index `irp_l1` in list 1 when used with reference index `irp_l0` in list 0 for bi-prediction.

`chroma_offset_bipred[irp_l0][irp_l1][1]` is the additive offset applied to the Cr prediction values when reference index `irp_l0` in list 0 is used with reference index `irp_l1` in list 1 for bi-prediction.

If `chroma_weight_flag_bipred[irp_l0][irp_l1]` is zero, `chroma_weight_bipred_l0[irp_l0][irp_l1][j]` and `chroma_weight_bipred_l1[irp_l0][irp_l1][j]` shall be interpreted as equal to $2^{\text{chroma_log_weight_denom}}$ and `chroma_offset_bipred[irp_l0][irp_l1][j]` shall be interpreted as equal to zero.

DRAFT ISO/IEC 14496-10 : 2002 (E)

For any combination of `irp_l0` and `irp_l1` that is not sent, the following values shall be inferred:

- `luma_weight_bipred_l0[irp_l0][irp_l1] = luma_weight_l0[irp_l0],`
- `luma_weight_bipred_l1[irp_l0][irp_l1] = luma_weight_l1[irp_l1],`
- `luma_offset_bipred[irp_l0][irp_l1] =`
 $(\text{luma_offset_l0[irp_l0]} + \text{luma_offset_l1[irp_l1]} - 1) \gg 1$
- `chroma_weight_bipred_l0[irp_l0][irp_l1][j] = chroma_weight_l0[irp_l0][j],`
- `chroma_weight_bipred_l1[irp_l0][irp_l1][j] = chroma_weight_l1[irp_l1][j],`
- `chroma_offset_bipred[irp_l0][irp_l1][j] =`
 $(\text{chroma_offset_l0[irp_l0][j]} + \text{chroma_offset_l1[irp_l1][j]} + 1) \gg 1$

7.4.4 Slice data semantics

mb_skip_run indicates a number of consecutive macroblocks decoded as MbSkip macroblock type in P slices or Direct_16x16 macroblock type and no additional transform coefficients in B slices when `entropy_coding_mode == 0`.

For `mb_frame_field_adaptive_flag == 1`, refer to the MB scanning order in Figure 6-4 (subclause 6.3).

mb_skip_flag indicates that the macroblock is decoded as MbSkip macroblock type in P slices or Direct_16x16 macroblock type and no additional transform coefficients in B slices when `entropy_coding_mode == 1`.

For the MbSkip macroblock type no further information about the macroblock is decoded. The motion vector for a MbSkip macroblock type shall be obtained as described in subclause 8.3.1.3. If `pic_structure` indicates a frame, then the decoded frame with `ref_idx_l0 == 0`, which either was decoded from a frame picture or is the union of two decoded field pictures shall be used as reference in motion compensation. If `pic_structure` indicates a field, then the decoded field of the same parity (top or bottom) with `ref_idx_l0 == 0`, which was either decoded from a field picture or is part of the most recently decoded frame picture shall be used as reference in motion compensation. For `mb_frame_field_adaptive_flag == 1`, a pair of macroblocks will be in frame mode only. If one of a pair of macroblocks is skipped, it should be in the same frame/field coding mode as another macroblock of the same pair. In field coding, the skipped macroblock is decoded by copying the co-located macroblock from the most recently decoded I or P field of the same field parity.

mb_field_decoding_flag equal to zero indicates that the macroblock pair is decoded in frame decoding mode and equal to one indicates that the macroblock pair is decoded in field decoding mode.

end_of_slice_flag equal to 0 indicates that another macroblock is following, whereas equal to 1 indicates the end of the slice and that no further macroblock follows. `end_of_slice_flag` is only present if `entropy_coding_mode == 1` for each macroblock except for the last macroblock of the picture.

If the current picture is a frame-structured picture and `mb_adaptive_frame_field_flag` is 1, the number of coded macroblocks in the slice shall be an even number.

7.4.5 Macroblock layer semantics

mb_type indicates the macroblock types. The semantics of `mb_type` depend on the slice type. In the following, for I slices, SI slices, P slices, SP slices, and B slices, tables and semantics are specified for the various macroblock types.

The macroblock types for I slices are specified in Table 7-10. If `adaptive_block_size_transform_flag == 1`, the macroblock types for I slices are specified in Table 12-1.

Table 7-10 – Macroblock types for I slices

Value of <code>mb_type</code>	Name of <code>mb_type</code> in I slices	<code>mb_partition_pred_mode(, 1)</code>	<code>num_sub_blocks()</code>	<code>num_mb_intra_partition()</code>
0	Intra_4x4	Intra	4	16
1	Intra_16x16_0_0_0	Intra	4	na
2	Intra_16x16_1_0_0	Intra	4	na
3	Intra_16x16_2_0_0	Intra	4	na
4	Intra_16x16_3_0_0	Intra	4	na
5	Intra_16x16_0_1_0	Intra	4	na
6	Intra_16x16_1_1_0	Intra	4	na

7	Intra_16x16_2_1_0	Intra	4	na
8	Intra_16x16_3_1_0	Intra	4	na
9	Intra_16x16_0_2_0	Intra	4	na
10	Intra_16x16_1_2_0	Intra	4	na
11	Intra_16x16_2_2_0	Intra	4	na
12	Intra_16x16_3_2_0	Intra	4	na
13	Intra_16x16_0_0_1	Intra	4	na
14	Intra_16x16_1_0_1	Intra	4	na
15	Intra_16x16_2_0_1	Intra	4	na
16	Intra_16x16_3_0_1	Intra	4	na
17	Intra_16x16_0_1_1	Intra	4	na
18	Intra_16x16_1_1_1	Intra	4	na
19	Intra_16x16_2_1_1	Intra	4	na
20	Intra_16x16_3_1_1	Intra	4	na
21	Intra_16x16_0_2_1	Intra	4	na
22	Intra_16x16_1_2_1	Intra	4	na
23	Intra_16x16_2_2_1	Intra	4	na
24	Intra_16x16_3_2_1	Intra	4	na

The following semantics are assigned to the macroblock types in Table 7-10:

- Intra_4x4: the macroblock is coded as Intra prediction type.
- Intra_16x16_x_y_z: x: Imode, y: nc, z: AC these modes refer to 16x16 Intra prediction type. Imode numbers from 6 and upwards represent 16x16 intra coding.

The macroblock types for SI slices are specified in Table 7-11 and Table 7-10. The mb_type value 0 is specified in Table 7-11 and the mb_type values 1 to 25 are specified in Table 7-10 by adding 1 to the value of mb_type in Table 7-10.

NOTE - If adaptive_block_size_transform_flag == 1, the use of SI slices is not allowed.

Table 7-11 – Macroblock type with value 0 for SI slices

Value of mb_type	Name of mb_type SI slices	mb_partition_pred_mode(, I)	num_sub_blocks()
0	SIIntra_4x4	SIIntra	4

The following semantics are assigned to the macroblock types in Table 7-11:

- SIIntra_4x4: the macroblock is coded as SIIntra prediction type.

The macroblock types for P slices are specified in Table 7-12 and Table 7-10. The mb_type values 0 to 4 are specified in Table 7-12 and the mb_type values 5 to 28 are specified in Table 7-10 by adding 5 to the value of mb_type in Table 7-10.

If adaptive_block_size_transform_flag == 1, the macroblock types for P slices are specified in Table 7-12 and Table 12-1. The mb_type values 0 to 4 are specified in Table 7-12 and the mb_type values 5 to 8 are specified in Table 12-1 by adding 5 to the value of mb_type in Table 12-1.

DRAFT ISO/IEC 14496-10 : 2002 (E)

The macroblock types for SP slices are specified in Table 7-12, Table 7-11, and Table 7-10. The mb_type values 0 to 4 are specified in Table 7-12, the mb_type value 5 is specified in Table 7-11 by adding 5 to the value of mb_type in Table 7-11, and the mb_type values 6 to 29 are specified in Table 7-10 by adding 6 to the value of mb_type in Table 7-10.

NOTE - If adaptive_block_size_transform_flag == 1, the use of SP slices is not allowed.

Table 7-12 – Macroblock type values 0 to 4 for P and SP slices

Value of mb_type	Name of mb_type	num_mb_partition()	mb_partition_pred_mode(, 1)	mb_partition_pred_mode(, 2)	num_sub_blocks()
0	Pred_L0_16x16	1	Pred_L0		4
1	Pred_L0_L0_16x8	2	Pred_L0	Pred_L0	4
2	Pred_L0_L0_8x16	2	Pred_L0	Pred_L0	4
3	Pred_8x8	4	Na	na	4
4	Pred_8x8ref0	4	Na	na	na
					na

The following semantics are assigned to the macroblock types in Table 7-12:

- Pred_L0_16x16, Pred_L0_L0_16x8, Pred_L0_L0_8x16, and Pred_8x8: the macroblock is predicted from a past picture with luma block sizes 16x16, 16x8, 8x16, and 8x8, respectively, and the associated chroma blocks. For the macroblock types NxM=16x16, 16x8, and 8x16, a motion vector is provided for each NxM luma block and the associated chroma blocks. If N=M=8, for each 8x8 sub macroblock an additional syntax element is decoded which indicates in which type the corresponding sub macroblock is decoded (see subclause 7.4.5.2). Depending on N,M and the sub macroblock types modes there may be 1 to 16 sets of motion data elements for a macroblock.
- Pred_8x8ref0: same as Pred_8x8 but ref_idx_l0 is not sent and set to 0 for all sub macroblocks.

The macroblock types for B slices are specified in Table 7-13 and Table 7-10. The mb_type values 0 to 22 are specified in Table 7-13 and the mb_type values 23 to 57 are specified in Table 7-10 by adding 23 to the value of mb_type in Table 7-10.

If adaptive_block_size_transform_flag == 1, the macroblock types for B slices are specified in Table 7-13 and Table 12-1. The mb_type values 0 to 22 are specified in Table 7-13 and the mb_type values 23 to 27 are specified in Table 12-1 by adding 23 to the value of mb_type in Table 12-1.

Table 7-13 – Macroblock type values 0 to 22 for B slices

Value of mb_type	Macroblock type mb_type name	num_mb_partition()	mb_partition_pred_mode(, 1)	mb_partition_pred_mode(, 2)	num_sub_blocks()
0	Direct_16x16	1	Direct		4
1	Pred_L0_16x16	1	Pred_L0		4
2	BiPred_L1_16x16	1	Pred_L1		4
3	BiPred_Bi_16x16	1	BiPred		4
4	Pred_L0_L0_16x8	2	Pred_L0	Pred_L0	4
5	Pred_L0_L0_8x16	2	Pred_L0	Pred_L0	4
6	BiPred_L1_L1_16x8	2	Pred_L1	Pred_L1	4
7	BiPred_L1_L1_8x16	2	Pred_L1	Pred_L1	4
8	BiPred_L0_L1_16x8	2	Pred_L0	Pred_L1	4
9	BiPred_L0_L1_8x16	2	Pred_L0	Pred_L1	4

10	BiPred_L1_L0_16x8	2	Pred_L1	Pred_L0	4
11	BiPred_L1_L0_8x16	2	Pred_L1	Pred_L0	4
12	BiPred_L0_Bi_16x8	2	Pred_L0	BiPred	4
13	BiPred_L0_Bi_8x16	2	Pred_L0	BiPred	4
14	BiPred_L1_Bi_16x8	2	Pred_L1	BiPred	4
15	BiPred_L1_Bi_8x16	2	Pred_L1	BiPred	4
16	BiPred_Bi_L0_16x8	2	BiPred	Pred_L0	4
17	BiPred_Bi_L0_8x16	2	BiPred	Pred_L0	4
18	BiPred_Bi_L1_16x8	2	BiPred	Pred_L1	4
19	BiPred_Bi_L1_8x16	2	BiPred	Pred_L1	4
20	BiPred_Bi_Bi_16x8	2	BiPred	BiPred	4
21	BiPred_Bi_Bi_8x16	2	BiPred	BiPred	4
22	BiPred_8x8	4	na		na

The following semantics are assigned to the macroblock types in Table 7-13:

- Direct_16x16 type: no motion vector data is transmitted.
- BiPred_x_y_NxM, x,y=L0,L1,Bi: each NxM block of a macroblock is predicted by using distinct motion vectors, reference pictures, and prediction directions. As indicated in Table 11-11, three different macroblock types that differ in their prediction methods exist for the 16x16 mode. For the 16x8 and 8x16 macroblock types, nine different combinations of the prediction methods are possible. If a macroblock is coded in 8x8 mode, an additional codeword for each 8x8 sub-partition indicates the decomposition of the 8x8 block as well as the chosen prediction direction (see Table 11-2).
- BiPred_8x8: the macroblock is partitioned into sub macroblocks. The coding of each sub macroblock is specified using sub_mb_type.

coded_block_pattern contains information which of the 8x8 blocks - luma and chroma - contain transform coefficients. An 8x8 block contains four 4x4 blocks. An indication that an 8x8 block contains coefficients means that one or more of the four 4x4 blocks within the 8x8 block contains coefficients*. The four least significant bits of coded_block_pattern contain information on which of four 8x8 luma blocks in the macroblock contains non-zero coefficients. These four bits are denoted as coded_block_patternY. The ordering of 8x8 blocks is indicated in Figure 6-5. A bit equal to zero in position n of coded_block_pattern (binary representation) indicates that the corresponding 8x8 block has no coefficients and a bit equal to 1 indicates that the 8x8 block has one or more non-zero coefficients. For chroma, 3 possibilities are specified in Table 7-14.

Table 7-14 – Specification of nc values

Value of nc	Description
0	All chroma coefficients are 0.
1	One ore more chroma DC coefficients are non-zero. All chroma AC coefficients are 0.
2	Zero or more chroma DC coefficients are non-zero. One ore more chroma AC coefficients are non-zero.

The value of coded_block_pattern for a macroblock is given by coded_block_pattern = coded_block_patternY + 16 x nc

The coded_block_pattern is indicated with a different codeword for macroblocks in P, SP, and B slices compared to macroblocks in I and SI slices.

If adaptive_block_size_transform_flag == 1, the semantics for coded_block_pattern are specified in subclause 12.3.

delta_qp the value of QP_Y can be changed in the macroblock layer by the parameter delta_qp. The delta_qp parameter is present only for non-skipped macroblocks, as defined by:

DRAFT ISO/IEC 14496-10 : 2002 (E)

- If coded_block_pattern indicates that there are nonzero transform coefficients in the macroblock or
- If the macroblock is 16x16 based intra coded

Furthermore, when macroblock-level adaptive frame/field coding is in use, delta_qp is present only for the first non-skipped macroblock of each macroblock pair.

The decoded value of delta_qp shall be in the range from -26 to +25, inclusive. This specifies the value of QP_Y in the range [0..51], as given by

$$QP_Y = (QP_Y + \text{delta_qp} + 52) \% 52 \quad (7-9)$$

7.4.5.1 Macroblock prediction semantics

All samples of the macroblock are predicted. The prediction mode is signalled using the mb_type syntax element. Depending on the mb_type syntax element, intra prediction information is signalled using intra_pred_mode or inter prediction information is signalled using the syntax elements ref_idx_10, ref_idx_11, mvd_10, and mvd_11.

If adaptive_block_size_transform_flag == 1, modifications to the macroblock prediction semantics are specified in subclause 12.2.1.1.

The return value of num_mb_intra_partition() is always equal to 16 if adaptive_block_size_transform_flag == 0.

NOTE - If adaptive_block_size_transform_flag == 1 the semantics regarding the function num_mb_intra_partition() are specified in subclause 12.2.1.1.

intra_pred_mode indicates the intra prediction mode.

intra_chroma_pred_mode indicates the type of spatial prediction used for chroma whenever any part of the luma macroblock is intra coded. This is shown in Table 7-15.

Table 7-15 – Relationship between intra_chroma_pred_mode and spatial prediction modes

Value intra_chroma_pred_mode	of Prediction Mode
0	DC
1	Horizontal
2	Vertical
3	Plane

ref_idx_10, when present, indicates the reference picture to be used for prediction.

~~ref_idx_10 if num_ref_idx_10_active_minus1 indicates the possibility of prediction from more than one reference picture, the exact picture to be used must be indicated. If pic_structure indicates that the current picture is a frame picture, then the reference picture is a previous frame in list 0 that was either indicated as a single frame picture or a frame that was indicated as two field pictures and has been reconstructed as a frame. Thus for frames the following table gives the reference frame:~~

Code_num	Reference frame
0	The first frame in list 0
1	The second frame in list 0
2	The third frame in list 0
..	..

If pic_structure indicates that the current picture is a field picture, then the reference picture is a previous field in list 0 that was either coded as part of a frame-structured picture or coded as a field-structured picture. Thus for fields the following table gives the reference field:

Code_num	Reference field
0	The first field in list 0

Formatted: Font: Not Bold

1 The second field in list 0

2 The third field in list 0

.. ..

If num_ref_idx_l0_active_minus1 is equal to 0, ref_idx_l0 is not present. If num_ref_idx_l0_active_minus1 is equal to 1, only a single encoded bit is used to represent ref_idx_l0. If num_ref_idx_l0_active_minus1 is greater than 1, the value of ref_idx_l0 is represented by a decoded index.

ref_idx_l1 has the same semantics as ref_idx_l0, except that it is applied to the reference index list 1.

~~[[Ed.Note: the text in brackets is inserted from B picture clause and needs to be improved]]The ref_idx_l0 parameter indicates the position of the reference picture in list 0 to be used for list 0 motion-compensated prediction of the current macroblock. The ref_idx_l0 parameter is transmitted for each 16x16, 16x8, 8x16 block, and each 8x8 sub-partition if list 0 or bi-predictive prediction is used and if multi-frame list 0 prediction is in use. ref_idx_l0 is present only when num_ref_idx_l0_active_minus1 is not zero (when the use of multiple list 0 reference pictures is indicated). The code num and interpretation for ref_idx_l0 differs for pic_structure indicating field picture or frame picture and is described in subclause 8.6.3.~~

~~Similarly, ref_idx_l1 is the index into list 1 to determine which frame or field is used for list 1 prediction when multi-frame list 1 prediction is in use. ref_idx_l1 is present only when num_ref_idx_l1_active_minus1 is not zero (when the use of multiple list 1 reference pictures is indicated). ref_idx_l1 is used in the same manner as ref_idx_l0, but is an index into list 1 rather than list 0, as described in subclause 8.6.3.~~

mvd_l0 if so indicated by mb_type, vector data for 1-16 blocks are transmitted. For every block a prediction is formed for the horizontal and vertical components of the motion vector. mvd_l0 signals the difference between the vector component to be used and this prediction. Motion vectors are allowed to point to samples outside the reference picture. If a sample outside the reference picture is referred to in the prediction process, the nearest sample belonging to the picture (an edge or corner sample) shall be used. All fractional sample positions shall be interpolated as described in subclause 8.3.2. If a sample referred in the interpolation process (necessarily integer accuracy) is outside of the reference picture it shall be replaced by the nearest sample belonging to the picture (an edge or corner sample). Reconstructed motion vectors shall be clipped to ± 19 integer samples outside of the picture.

mvd_l1 has the same semantics as mvd_l0, except that it is applied to the reference index list 1.

7.4.5.2 Sub macroblock prediction semantics

sub_mb_type indicates the sub macroblock types.

If adaptive_block_size_transform_flag == 1, modifications to the sub macroblock prediction semantics are specified in subclause 12.2.1.2.

The return value of num_sub_mb_intra_partition() is always equal to 4 if adaptive_block_size_transform_flag == 0.

NOTE - If adaptive_block_size_transform_flag == 1 the semantics regarding the function num_sub_mb_intra_partition() are specified in subclause 12.2.1.1.

The sub macroblock types for P macroblocks are defined in Table 7-16.

Table 7-16 – Sub macroblock types in P macroblocks

Value of sub_mb_type	Name of sub_mb_type	num_sub_mb_partition()	sub_mb_pred_mode()	num_sub_mb_intra_partition()	num_sub_blocks ()
0	Pred_L0_8x8	1	Pred_L0	na	4
1	Pred_L0_8x4	2	Pred_L0	na	4
2	Pred_L0_4x8	2	Pred_L0	na	4
3	Pred_L0_4x4	4	Pred_L0	4	4
4	Intra_8x8	na	Intra		

The following semantics are assigned to the sub macroblock types in Table 7-16:

- Pred_L0_XxY, X,Y=4,8 the corresponding partition of the sub macroblock is predicted from a past picture with luma block size 8x4, 4x8, and 4x4, respectively, and the associated chroma blocks. A motion vector is

DRAFT ISO/IEC 14496-10 : 2002 (E)

transmitted for each $N \times M = 8 \times 4$, 4×8 , and 4×4 block. Depending on N and M , up to 4 motion vectors may be decoded for a sub macroblock, and thus up to 16 motion vectors maybe dedoced for a macroblock.

- Intra_8x8 the 8×8 sub-partition is coded in intra mode. Intra_8x8 *shall not* be present in SPred slices.

The sub macroblock types for B macroblocks are defined in Table 7-17.

Table 7-17 – Sub macroblock types in B macroblocks

Value of sub_mb_type	Name of sub_mb_type	num_sub_mb_partition()	sub_mb_pred_mode()	num_sub_mb_intra_partition()	num_sub_blocks()
0	Direct_8x8	1	Direct	na	4
1	Pred_L0_8x8	1	Pred_L0	na	4
2	BiPred_L1_8x8	1	Pred_L1	na	4
3	BiPred_Bi_8x8	1	BiPred	na	4
4	Pred_L0_8x4	2	Pred_L0	na	4
5	Pred_L0_4x8	2	Pred_L0	na	4
6	BiPred_L1_8x4	2	Pred_L1	na	4
7	BiPred_L1_4x8	2	Pred_L1	na	4
8	BiPred_Bi_8x4	2	BiPred	na	4
9	BiPred_Bi_4x8	2	BiPred	na	4
10	Pred_L0_4x4	4	Pred_L0	na	4
11	BiPred_L1_4x4	4	Pred_L1	na	4
12	BiPred_Bi_4x4	4	BiPred	na	4
13	Intra_8x8	1	Intra	4	4

The following semantics are assigned to the sub macroblock types in Table 7-17:

- Pred_L0_XxY, $X, Y = 4, 8$, have the same semantics as in Table 7-16.
- BiPred_Z_X_Y, $Z = L1, Bi$, $X, Y = 4, 8$
- Intra_8x8 has the same semantics as in Table 7-16.

7.4.5.3 Residual data semantics

If adaptive_block_size_transform_flag == 1, modifications to the residual data semantics are specified in subclause 12.2.1.3.

The return value of num_sub_blocks() is always equal to 4 if adaptive_block_size_transform_flag == 0.

NOTE - If adaptive_block_size_transform_flag == 1 the semantics regarding the function num_sub_blocks() are specified in subclause 12.2.1.3.

7.4.5.3.1 Residual 4x4 block CAVLC semantics

7.4.5.3.2 Residual 4x4 block CABAC semantics

coded_block_flag indicates whether the block contains non-zero transform coefficients. If coded_block_flag is equal to 0, the block contains no non-zero transform coefficients. If coded_block_flag is equal to 1, the block contains at least one non-zero transform coefficient.

significant_coeff_flag[i] indicates whether the transform coefficient at scanning position i is non-zero. If significant_coeff_flag[i] is equal to 0, the transform coefficient at scanning position i is equal to zero; if significant_coeff_flag[i] is equal to 1, the transform coefficient at position i has a non-zero value.

last_significant_coeff_flag[i] indicates for the scanning position *i* whether there are non-zero transform coefficients for subsequent scanning positions *i*+1 to max_numcoeff-1. If all following transform coefficients (in scanning order) of the block have value equal to zero last_significant_coeff_flag[i] is equal to 1. If last_significant_coeff_flag[i] is equal to 0, there are further non-zero transform coefficients along the scanning path.

coeff_absolute_value_minus_1 is the absolute value of the transform coefficient minus 1.

coeff_sign is the sign of the transform coefficient. A coeff_sign equal to 0 indicates a positive transform coefficient; a coeff_sign equal to 1 indicates a negative transform coefficient.

8 Decoding process

8.1 Ordering of decoding process

A macroblock or sub-partition is decoded in the following order.

1. Parsing of syntax elements using VLC/CAVLC (see subclause 9.1) or CABAC (see subclause 9.2)
2. Motion compensation (see subclause 8.4) or Intra prediction (see subclause 8.5)
3. Transform coefficient decoding (see subclause 8.6)
4. Deblocking Filter (see subclause 8.7)

8.2 NAL unit decoding

8.2.1 NAL unit delivery and decoding order

This subclause presents the requirements for the NAL unit delivery and decoding order.

Decoders conforming to this Recommendation | International Standard shall be capable of receiving NAL units in decoding order. Systems conveying NAL unit streams conforming to this Recommendation | International Standard shall either

- 1) Present NAL unit streams to the decoder in decoding order, or
- 2) Provide a means to indicate the NAL unit decoding order to the decoder in the case of enhanced-capability decoders which may be capable of receiving or processing some NAL units in an out-of-order fashion. No such enhanced capability is defined or required herein for decoders conforming to this Recommendation | International Standard.

The decoding order of a sequence parameter set shall precede the decoding order of other NAL units that refer to that sequence parameter set.

The decoding order of a picture parameter set shall precede the decoding order of other NAL units that refer to that picture parameter set.

A coded picture is called a primary coded picture if the redundant_slice_flag is 0 or if its slice headers contain redundant_pic_cnt equal to 0.

The decoding order of coded slices and data partitions of a primary coded picture shall be contiguous relative to the decoding order of coded slices and data partitions of other primary coded pictures.

The decoding order of coded slices and data partitions of a primary or redundant coded picture shall precede the decoding order of coded slices and data partitions of any other coded picture that uses the primary coded picture as a reference for inter-picture prediction.

The decoding order of any coded slices or data partitions of a primary coded picture shall precede the decoding order of any redundant slices or data partitions containing coded data for the same macroblock locations represented in the slice or data partition for the primary decoded picture.

The decoding order of any redundant coded slice or data partition shall precede the decoding order of the coded slices and data partitions of any other coded picture that uses the primary coded picture corresponding to the coded slice or data partition of the redundant coded slice or data partition as a reference for inter-picture prediction.

The decoding order of slices and data partitions of primary coded pictures shall be non-decreasing in frame number order. The decoding order of the slices and data partitions of non-stored primary coded pictures shall be subsequent to the decoding order of the slices and data partitions of the stored picture with the same frame number. If multiple

DRAFT ISO/IEC 14496-10 : 2002 (E)

primary coded pictures share the same frame number, the decoding order of the slices and data partitions of the non-stored pictures shall be in ascending order of redundant pic cnt.

Depending on the profile in use, arbitrary slice ordering may or may not be allowed. If arbitrary slice ordering is allowed, the slices and data partitions of a primary coded picture may follow any decoding order relative to each other. If arbitrary slice ordering is not allowed, the decoding order of slices and data partitions of a primary coded picture shall be increasing in the raster scan order of the first macroblock of each slice, and the decoding order of data partition A of a coded slice shall precede the decoding order of data partition B of the same coded slice, and the decoding of data partition B of a coded slice shall precede the decoding order of data partition C of the same coded slice, and data partitions A, B, and C of a coded slice shall be contiguous in decoding order relative to the decoding order of any data partitions or non-partitioned slice data NAL units of other coded slices.

The decoding order of SEI NAL units shall precede the decoding order of the slices and the slices and data partitions of the corresponding slice or data partition or coded picture or sequence of pictures to which the SEI NAL unit corresponds, and shall be subsequent to the decoding order of any SEI NAL units, slices, and data partitions of pictures that precede the corresponding coded picture in decoding order.

The decoding order of a picture delimiter, if present, shall precede the decoding order of the SEI NAL units, slices and data partitions of the corresponding coded picture, and shall be subsequent to the decoding order of any SEI NAL units, slices, and data partitions of pictures that precede the corresponding coded picture in decoding order.

8.2.2 Parameter set decoding

The decoder maintains 16 sequence parameter set locations. For each 16 possible values of seq_parameter_set_id, the most recent decoded sequence parameter set is copied into the location referenced by seq_parameter_set_id immediately before the decoding of the next IDR picture.

The decoder maintains 64 picture parameter set locations. For each 64 possible values of pic_parameter_set_id, the most recent picture parameter set is copied into the location referenced by pic_parameter_set_id immediately before the decoding of a slice or DPA belonging to the next coded picture.

8.3 Slice decoding

8.3.1 Detection of coded picture boundaries

Decoding of a new picture is started from the slice to be decoded, herein called the current slice, if the slice is not a redundant slice and if one of the following conditions is true:

1. The frame number of the current slice is different from the frame number of the previously decoded slice.
2. The frame number of the current slice is the same as frame number of the previously decoded slice, the nal_storage_idc of the previously decoded slice is zero, and the nal_storage_idc of the current slice is non-zero.
3. The frame number of the current slice is the same as frame number of the previously decoded slice, pic_order_cnt_type is 0, and pic_order_cnt is different from the pic_order_cnt in the previously decoded slice.
4. The frame number of the current slice is the same as the previously decoded slice, pic_order_cnt_type is 1 and delta_pic_order_cnt is different from the delta_pic_order_cnt in the previously decoded slice.

8.3.2 Picture order count

Each coded picture is associated with a picture order count, called PicOrderCnt, which is a 32-bit signed integer. An IDR picture shall have PicOrderCnt equal to 0. The PicOrderCnt of each stored picture shall be stored as long as the picture stays in the reference picture buffer.

The decoder should treat a wraparound, underflow or overflow of pic_order_cnt, PicOrderCntOffset, FrameNumOffset and AbsFrameNum defined in subclauses 8.3.2.1 and 8.3.2.2 as an error.

8.3.2.1 Picture order count type 0

If pic_order_cnt_type is 0, the decoder shall maintain a picture order count offset, called PicOrderCntOffset, which is a 32-bit signed integer. The PicOrderCntOffset shall be zero for an IDR picture.

If pic_order_cnt_type is 0 and if the decoding of a new picture is started, the PicOrderCntOffset is updated and the pic_order_cnt of the picture is calculated. The pic_order_cnt of the previous picture in decoding order is herein called PreviousPicOrderCnt. If pic_order_cnt is smaller than PreviousPicOrderCnt and if (PreviousPicOrderCnt - pic_order_cnt) is greater than or equal to (MAX_PIC_ORDER_CNT / 2), PicOrderCntOffset is calculated according to Equation 8-1.

$$\text{PicOrderCntOffset} = \text{PicOrderCntOffset} + \text{MAX_PIC_ORDER_CNT} \quad (8-1)$$

If `pic_order_cnt` is greater than `PreviousPicOrderCnt` and if $(\text{pic_order_cnt} - \text{PreviousPicOrderCnt})$ is greater than or equal to $(\text{MAX_PIC_ORDER_CNT} / 2)$, `PicOrderCntOffset` is calculated according to Equation 8-2.

$$\text{PicOrderCntOffset} = \text{PicOrderCntOffset} - \text{MAX_PIC_ORDER_CNT} \quad (8-2)$$

Otherwise, the value of `PicOrderCntOffset` is not changed.

If `pic_order_cnt_type` is 0 and if the decoding of a new picture is started, the `PicOrderCnt` of the picture is calculated according to Equation 8-3.

$$\text{PicOrderCnt} = \text{PicOrderCntOffset} + \text{pic_order_cnt} \quad (8-3)$$

8.3.2.2 Picture order count type 1

If `pic_order_cnt_type` is 1, the decoder shall maintain a `FrameNumOffset` that is a 32-bit unsigned integer. The `FrameNumOffset` of an IDR picture shall be zero.

If `pic_order_cnt_type` is 1 and if the decoding of a new picture is started, the `PicOrderCnt` of the picture is calculated. In the following, `AbsFrameNum` and `PicOrderCntCycleCount` are 32-bit unsigned integers, and the `frame_num` of the previous picture in decoding order is called `PreviousFrameNum`. First, the `FrameNumOffset` is updated. If `frame_num` is greater than or equal to the `PreviousFrameNum`, `FrameNumOffset` is unchanged. Otherwise, when `frame_num` is smaller than `PreviousFrameNum`, `FrameNumOffset` is calculated according to Equation 8-4.

$$\text{FrameNumOffset} = \text{FrameNumOffset} + \text{MAX_FN} \quad (8-4)$$

Second, the `AbsFrameNum` is calculated according to Equation 8-5.

$$\text{AbsFrameNum} = \text{FrameNumOffset} + \text{frame_num} \quad (8-5)$$

Third, the `PicOrderCntCycleCount` is calculated according to Equation 8-6.

$$\text{PicOrderCntCycleCount} = \text{AbsFrameNum} / \text{num_stored_frames_in_pic_order_cnt_cycle} \quad (8-6)$$

Fourth, the `FrameNumInPicOrderCntCycle` is calculated according to Equation 8-7.

$$\text{FrameNumInPicOrderCntCycle} = \text{AbsFrameNum} \% \text{num_stored_frames_in_pic_order_cnt_cycle} \quad (8-7)$$

In the following, the `EXPECTED_DELTA_PER_PIC_ORDER_CNT_CYCLE` is the sum of `offset_for_stored_frame` values. Finally, the `PicOrderCnt` of the picture is computed using the algorithm expressed in Equation 8-8.

$$\begin{aligned} \text{PicOrderCnt} &= \text{PicOrderCntCycleCount} \times \text{EXPECTED_DELTA_PER_PIC_ORDER_CNT_CYCLE} \\ \text{for } (i = 0; i \leq \text{FrameNumInPicOrderCntCycle}; i++) \\ &\quad \text{PicOrderCnt} = \text{PicOrderCnt} + \text{offset_for_stored_frame}_i \\ \text{PicOrderCnt} &= \text{PicOrderCnt} + \text{delta_pic_order_cnt} \\ \text{if } (\text{nal_storage_idx} == 0) \\ &\quad \text{PicOrderCnt} = \text{PicOrderCnt} + \text{offset_for_non_stored_pic} \end{aligned} \quad (8-8)$$

8.3.3 Decoder process for redundant slices

If the `redundant_pic_cnt` in the slice header of a coded slice is greater than 0, the decoder may discard the coded slice. If some of the samples in the decoded primary picture are incorrect and if the coded redundant slice can be correctly decoded, the decoder should replace the incorrect samples with the corresponding samples of the redundant decoded slice.

8.3.4 Specification of macroblock allocation map

If decoding of a new picture is started, if `num_slice_groups_minus1` is greater than 0 and if `mb_allocation_map_type` is 4, 5 or 6, a macroblock allocation map is generated. Slice group 0 has a growth order specified in subclauses 8.3.4.1-8.3.4.4. The number of macroblocks in slice group 0 is equal to `slice_group_change_cycle` × `SLICE_GROUP_CHANGE_RATE`. This number of macroblock locations in the specified growth order is allocated for slice group 0. The rest of the macroblocks of the picture are allocated for slice group 1.

8.3.4.1 Allocation order for box-out

Let H denote the number of coded macroblock rows of the picture and W denote the number of coded macroblock columns of the picture. Macroblock locations are indicated with coordinates (x, y) , where the top-left macroblock location of the picture has coordinates $(0, 0)$ and the bottom-right macroblock location has coordinates $(W - 1, H - 1)$. The allocation order is created using a `AllocationDirection` variable that indicates the next macroblock location relative to the current one. `AllocationDirection` can have four values: $(-1, 0)$, $(1, 0)$, $(0, -1)$ and $(0, 1)$. Furthermore, the left-most and right-most macroblock columns allocated in the allocation order and the top-most and bottom-most macroblock rows allocated in the allocation order are stored in the variables `Left`, `Right`, `Top`, and `Bottom` respectively. For the box-out clockwise macroblock allocation map type, the first macroblock location in the allocation order is $(x, y) = (W/2, H/2)$ and the initial `AllocationDirection` is $(-1, 0)$. For the counter-clockwise macroblock allocation map type, the first macroblock location in allocation order is $(x, y) = ((W - 1)/2, (H - 1)/2)$ and the initial `AllocationDirection` is $(0, 1)$. At the beginning, `Left` = `Right` = x , and `Top` = `Bottom` = y . A subsequent macroblock location (x, y) in allocation order is allocated by searching the first row from top to bottom in Table 8-1 for the same value of `AllocationDirection` and where the given condition is true. Then, the x , y , `AllocationDirection`, `Left`, `Right`, `Top` and `Bottom` variables are updated according to the refined macroblock allocation map type. If `Left` ≥ 0 , `Right` $< W$, `Top` ≥ 0 and `Bottom` $< H$, the next macroblock location (x, y) in allocation order is allocated as described above. Otherwise, all macroblock locations have been allocated.

Table 8-1 – Allocation order for the box-out macroblock map allocation type

AllocationDirection	Condition	Box-out clockwise	Box-out counter-clockwise
$(-1, 0)$	$x > \text{Left}$	$x = x - 1$	$x = x - 1$
$(-1, 0)$	$x == 0$	$y = \text{Top} - 1$ $\text{Top} = \text{Top} - 1$ $\text{AllocationDirection} = (1, 0)$	$y = \text{Bottom} + 1$ $\text{Bottom} = \text{Bottom} + 1$ $\text{AllocationDirection} = (1, 0)$
$(-1, 0)$	$x == \text{Left}$	$x = x - 1$ $\text{Left} = \text{Left} - 1$ $\text{AllocationDirection} = (0, -1)$	$x = x - 1$ $\text{Left} = \text{Left} - 1$ $\text{AllocationDirection} = (0, 1)$
$(1, 0)$	$x < \text{Right}$	$x = x + 1$	$x = x + 1$
$(1, 0)$	$x == W - 1$	$y = \text{Bottom} + 1$ $\text{Bottom} = \text{Bottom} + 1$ $\text{AllocationDirection} = (-1, 0)$	$y = \text{Top} - 1$ $\text{Top} = \text{Top} - 1$ $\text{AllocationDirection} = (-1, 0)$
$(1, 0)$	$x == \text{Right}$	$x = x + 1$ $\text{Right} = \text{Right} + 1$ $\text{AllocationDirection} = (0, 1)$	$x = x + 1$ $\text{Right} = \text{Right} + 1$ $\text{AllocationDirection} = (0, -1)$
$(0, -1)$	$y > \text{Top}$	$y = y - 1$	$y = y - 1$
$(0, -1)$	$y == 0$	$x = \text{Right} + 1$ $\text{Right} = \text{Right} + 1$ $\text{AllocationDirection} = (0, 1)$	$x = \text{Left} - 1$ $\text{Left} = \text{Left} - 1$ $\text{AllocationDirection} = (0, 1)$
$(0, -1)$	$y == \text{Top}$	$y = y - 1$ $\text{Top} = \text{Top} - 1$ $\text{AllocationDirection} = (1, 0)$	$y = y - 1$ $\text{Top} = \text{Top} - 1$ $\text{AllocationDirection} = (-1, 0)$
$(0, 1)$	$y < \text{Bottom}$	$y = y + 1$	$y = y + 1$
$(0, 1)$	$y == H - 1$	$x = \text{Left} - 1$ $\text{Left} = \text{Left} - 1$ $\text{AllocationDirection} = (0, -1)$	$x = \text{Right} + 1$ $\text{Right} = \text{Right} + 1$ $\text{AllocationDirection} = (0, -1)$